ARM

Deploying a Caffe MNIST model using the Arm NN SDK

Document number:	ARMECM-21-1342	Version:	1.0
Date of Issue:	18/04/2018		

Abstract

Learn how to get your Caffe application running with the Arm NN SDK.

Keywords

Machine learning, artificial intelligence, neural network, Arm NN SDK, Compute Library

Contents

1 Overview

Using an example application, this guide shows you how to run a Caffe model using the open-source Arm NN SDK. You will be able to use the knowledge you gain from this guide to run your own models on Arm Cortex CPUs and Mali GPUs.

2 Prerequisites

This guide assumes:

- You are familiar with neural networks and the MNIST dataset. If you are new to either of these concepts, read this Caffe introduction and this overview of MNIST.
- You have access to the complete example application with source code, data, and model.

3 Building your Caffe model using the Arm NN SDK

To run a Caffe model using the Arm NN SDK:

- Load and parse the MNIST test set.
- Import a graph.
- Optimize and load onto a compute device.
- Run a graph on a device.

Using the example code, this guide walks you through each step.

3.1 Load and parse the MNIST test set

To begin building your own Caffe model, load and parse the MNIST test set.

This example code loads and parses the MNIST test set:

```
// Load a test image and its correct label
std::string dataDir = "data/";
int testImageIndex = 0;
std::unique_ptr input = loadMnistImage(dataDir, testImageIndex);
```

3.2 Import a graph

The Arm NN SDK provides parsers for reading model files from Caffe.

The SDK supports Caffe graphs in text and binary ProtoBuf formats. To import the graph:

- Load the model.
- Bind the input and output points of its graph.

This example code imports the graph:

```
// Import the Caffe model. Note: use CreateNetworkFromTextFile for text
files.
armnnCaffeParser::ICaffeParserPtr parser =
armnnCaffeParser::ICaffeParser::Create();
```

```
armnn::INetworkPtr network =
    parser->CreateNetworkFromBinaryFile("model/lenet_iter_9000.caffemodel",
        { }, // input taken from file if empty
        { "prob" }); // output node
```

After this step, the code is common regardless of the framework that you started with.

This example code binds the input and output tensors to the data and selects the loaded network identifier:

```
// Find the binding points for the input and output nodes
armnnCaffeParser::BindingPointInfo inputBindingInfo =
    parser->GetNetworkInputBindingInfo("data");
armnnCaffeParser::BindingPointInfo outputBindingInfo =
    parser->GetNetworkOutputBindingInfo("prob");
```

You can read the result of the inference from the output array and compare it to the MnistImage label from the data file.

3.3 Optimize and load onto a compute device

You must optimize your network and load it onto a compute device. The Arm NN SDK supports optimized execution on multiple CPU and GPU devices. Before you start executing a graph, you must select the appropriate device context and optimize the graph for that device.

You specify a preferential order of compute devices you want workloads to run on when you call the optimize() function. The optimizer attempts to schedule the workloads on the first specified device and falls back to the following device if the first is not available or does not support the workload. The following are the compute devices you specify to run workloads:

- The Arm Mali GPU which you use GpuAcc to specify.
- The Armv7 or Armv8 CPU which you use CpuAcc to specify.

This example code optimizes and loads your network onto a compute device:

```
//// Create a context and optimize the network for one or more compute
devices in order of preference
// e.g. GpuAcc, CpuAcc = if available run on Arm Mali GPU, else try to run
on ARM v7 or v8 CPU
armnn::IRuntime::CreationOptions options;
armnn::IRuntimePtr context = armnn::IRuntime::Create(options);
armnn::IOptimizedNetworkPtr optNet = armnn::Optimize(*net,
{armnn::Compute::GpuAcc, armnn::Compute::CpuAcc}, runtime->GetDeviceSpec());

// Load the optimized network onto the device
armnn::NetworkId networkIdentifier;
context->LoadNetwork(networkIdentifier, std::move(optNet));
```

3.4 Run a graph on a compute device

Inference on a compute device is performed using the `EnqueueWorkload()` function of the context.

This example code runs a single inference on the test image:

```
// Run a single inference on the test image
std::array<float, 10=""> output;
armnn::Status ret = context->EnqueueWorkload(networkIdentifier,
    MakeInputTensors(inputBindingInfo, &input->image[0]),
    MakeOutputTensors(outputBindingInfo, &output[0]));
```

The `std::distance()` function in the following example code is used to find the index of the largest element in the output. This function is equivalent to NumPy's `argmax()` function.

```
// Convert 1-hot output to an integer label and print
int label = std::distance(output.begin(),
std::max_element(output.begin(),output.end()));
std::cout << "Predicted: " << label << std::endl;
std::cout << "   Actual: " << input->label << std::endl;
return 0;
```

4 Deploying an application using the Arm NN SDK

You must link your application with the Arm NN SDK library and the Caffe parsing library. This example code links your application:

```
g++ -O3 -std=c++14 -I$(ARMNN_INC) mnist_caffe.cpp -o mnist_caffe -
L$(ARMNN_LIB)-larmnn -larmnnCaffeParser
```

For convenience, the Arm NN SDK can run with a reference implementation on x86 for development and testing, but the optimized kernels are only available for Arm CPUs and GPUs. This means that you must run your profiling and optimization steps on a real Arm-powered device because x86 performance does not represent real performance.